2 Motivating Example: Factorial

- The factorial of a non-negative integer n is

$$n! = \begin{cases} 1, & n = 0 \text{ or } n = 1 \\ n * (n-1) * \cdots * 1, & n > 1 \end{cases}$$

- For example:

$$5! = 5 * 4 * 3 * 2 * 1 = 120$$

$$1! = 1$$

$$0! = 1$$

① Factorial 数学概念.

3 Implementing Factorial

- To implement factorial, we could use a loop…

```
// REQUIRES: n >= 0
// EFFECTS: computes and returns n!
int fact(int n) {
  int result = 1;
  while (n > 0) {
    result *= n;
    --n;
  }
  return result;
}
```

- Question: Can we do it without a loop?

① while loop 实现 Factorial.

4 Recursive Factorial

- Can we do it without a loop, using **recursion** instead?

```
// REQUIRES: n >= 0
// EFFECTS: computes and returns n!
int fact(int n) {
  if (n == 0) {
    return 1;
  }
  else {
    return n * fact(n - 1);
  }
}
```

A recursive function calls itself to implement repetition.

L18.2_fact on Lobster

- Does this work?
  - How does one call set the parameters for the next?
  - How does the recursion know to stop?
  - Where does the multiplication happen?

6/7/2022

① recursion 实现 factorial.

5 Solving Problems with Recursion

- A recursive approach needs:

1. A base case
   - Can be solved without recursion
2. Recursive cases:
   - Break down into **subproblems** that are **similar** and **smaller** (closer to a base case)

fact(4) fact(3)

① Recursion 由两部分组成. base case + recursion case

② 使用 Recursion 的前提是：可分解出 subqestion. 且 subquestion 满足 similar 和 smaller (→ base case) 的特点...

6 No base case? Infinite Recursion!

Subproblems that are:

- Similar? YES
- Smaller? NO

```
void countToInfinity(int x) {
  cout << x << endl;
  // Recursive Case:
  countToInfinity(x + 1);
}

int main() {
  countToInfinity(0);
}
```

① 违反 subquestion smaller 的例子

7 Recurrence Relations

- Earlier, we defined factorial iteratively:

$$n! = \begin{cases} 1, & n = 0 \text{ or } n = 1 \\ n * (n-1) * \cdots * 1, & n > 1 \end{cases}$$

- But factorial can also be defined **recursively**:

$$n! = \begin{cases} 1, & n = 0 \text{ or } n = 1 \\ n * (n-1)!, & n > 1 \end{cases}$$

similar smaller

- Factorial is defined in terms of itself!
- This is called a **recurrence relation**.

① 数学可以表达 Recursion 的 base case 和 Recursion case 以及 smaller & similar 的特点.

8 Writing Recursive Functions

- Start with the recurrence relation:

Base Case

$$n! = \begin{cases} 1, & n = 0 \text{ or } n = 1 \\ n * (n-1)!, & n > 1 \end{cases}$$

Recursive Case

- This often translates directly to code:

```
// REQUIRES: n >= 0
// EFFECTS: computes and returns n!
int fact(int n) {
  if (n <= 1) { // BASE CASE
    return 1;
  }
  else {
    return n * fact(n - 1); // RECURSIVE CASE
  }
}
```

① 公式向 Coding 转变的方法.

9 The Recursive Leap of Faith

- Here's the trick to writing recursive functions…
- Use the function to write itself. Even before it's "done".
  - It's ok. Trust us.
- Check the "combination" step.

```
// REQUIRES: n >= 0
// EFFECTS: computes and returns n!
int fact(int n) {
  if (n <= 1) { // BASE CASE
    return 1;
  }
  else {
    return n * fact(n - 1); // RECURSIVE CASE
  }
}
```

X + correct for x-1

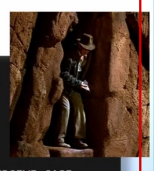


Image: Indiana Jones and The Last Crusade

6/7/2022

10 Exercise: Ducks

- Let's say we want to start a duck farm:
  - We start with 5 baby ducklings.
  - At age 1 month, and **every month** thereafter, each duck lays **3 eggs**.
  - An egg takes 1 month to hatch.
  - All eggs hatch, and ducks never die.
- How many ducks do we have after n months?

n	0	1	2	3	4	5	…
numDucks(n)	5	5	20	35	95	200	…

- Find a recurrence relation for numDucks(n).
  - Hint: The number of ducks is the number of previous ducks plus the number that have just hatched. (Two subproblems!)

6/7/2022

① 鸭子问题. ② 先总结表格规律，列出关系式，再找出 Base case，最终转化为 coding.

11 Exercise: Ducks

Question
How many Ducks do we have at month n=6?
A) 425 B) 460 C) 485 D) 755

- Given this recurrence relation:

```
numDucks(n) = numDucks(n-1) + 3 * numDucks(n-2)
```

Number of existing ducks. | Number of newly hatched ducks.

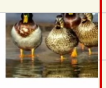
- Write code to implement numDucks:

```
// REQUIRES: n >= 0
// EFFECTS: computes the number of ducks at month n
int numDucks(int n) {

}
```

lobster.eecs.umich.edu ("L18.5_numDucks") 6/7/2022

12 Solution: Ducks



- Given this recurrence relation:

```
numDucks(n) = numDucks(n-1) + 3 * numDucks(n-2)
```

因为我的计算表达式中有n-2所以进入改表达式的n必须要大于或等于2. 所以这就是为什么base case中要设定n=0和1的时候都return 5

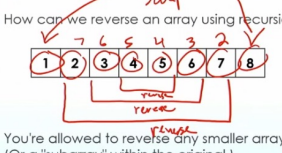
Number of existing ducks. | Number of newly hatched ducks.

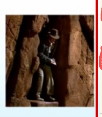
- Write code to implement numDucks:

```
// REQUIRES: n >= 0
// EFFECTS: computes the number of ducks at month n
int numDucks(int n) {
  if (n <= 1) { // BASE CASE
    return 5;
  }
  else {
    // RECURSIVE CASE
    return numDucks(n - 1) + 3 * numDucks(n - 2);
  }
}
```

13 Exercise: Recursive Reverse

- How can we reverse an array using recursion?





- You're allowed to reverse any smaller array. (Or a "subarray" within the original.)
- Brainstorm an algorithm and write pseudocode for it.
  - What's the recurrence?
  - What's the base case?

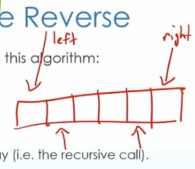
① Reverse 的例子.

② 以此是 Base case.

③ 以此是 Recursive case.

14 Exercise: Recursive Reverse

- Let's assume you're working with this algorithm:
- If the array size <= 0:
  1. Do nothing.
- If the array is not empty:
  1. Reverse the "middle" of the array (i.e. the recursive call).
  2. Swap the first/last elements of the array.




```
// EFFECTS: Reverses the array starting at 'left'
//          and ending at (and including) 'right'.
void reverse(int *left, int *right) {

}
```

lobster.eecs.umich.edu ("L18.6_reverse")

15 Solution: Recursive Reverse

- Let's assume you're working with this algorithm:
- If the array size <= 0:
  1. Do nothing.
- If the array is not empty:
  1. Reverse the "middle" of the array (i.e. the recursive call)
  2. Swap the first/last elements of the array.

```
// EFFECTS: Reverses the array starting at 'left'
//          and ending at (and including) 'right'.
void reverse(int *left, int *right) {
  if (left < right) {
    reverse(left + 1, right - 1);
    int temp = *left;
    *left = *right;
    *right = temp;
  }
}
```

① 先 reverse() recursion，直到 Base case，然后利用 Passive flow 逐一从中间两侧打印的 implementation.

20 Minute Exercise: The Cost of Recursion

- Let's say we want to reverse an array of size N
- For this **recursive** implementation of reverse…
  - **Time**: Find the number of swaps performed.
  - **Memory**: Find the *largest* number of stack frames for reverse on the stack at any given time.

```
// EFFECTS: Reverses the array starting at 'left'
//          and ending at (and including) 'right'.
void reverse(int *left, int *right) {
  if (left < right) {
    reverse(left + 1, right - 1);
    int temp = *left;
    *left = *right;
    *right = temp;
  }
}
```

① 在该方式 Time 和 Memory 与n相关.

21 Solution: The Cost of Recursion

- Let's say we want to reverse an array of size N
- For this **recursive** implementation of reverse…
  - **N/2** swaps are performed. **O(N) - linear time**
  - **N/2+1** stack frames are needed. **O(N) - linear space**

```
// EFFECTS: Reverses the array starting at 'left'
//          and ending at (and including) 'right'.
void reverse(int *left, int *right) {
  if (left < right) {
    reverse(left + 1, right - 1);
    int temp = *left;
    *left = *right;
    *right = temp;
  }
}
```

② Time 和 Space 和被交换的 element n之间的关系.

22 Minute Exercise: The Cost of Iteration

- Let's say we want to reverse an array of size N
- For this **iterative** implementation of reverse…
  - **Time**: Find the number of swaps performed.
  - **Memory**: Find the *largest* number of stack frames for reverse on the stack at any given time.

```
// EFFECTS: Reverses the array starting at 'left'
//          and ending at (and including) 'right'.
void reverse(int *left, int *right) {
  while (left < right) {
    int temp = *left;
    *left = *right;
    *right = temp;
    ++left;
    --right;
  }
}
```

① while loop 所被要用的 time 和 Memory.

23 Solution: The Cost of Iteration

- Let's say we want to reverse an array of size N
- For this **iterative** implementation of reverse…
  - **N/2** swaps are performed. **O(N) - linear time**
  - **1** stack frame is needed. **O(1) - constant space**

```
// EFFECTS: Reverses the array starting at 'left'
//          and ending at (and including) 'right'.
void reverse(int *left, int *right) {
  while (left < right) {
    int temp = *left;
    *left = *right;
    *right = temp;
    ++left;
    --right;
  }
}
```

6/7/2022

24 Tail Recursion

- Consider these two implementations of reverse:

lots of memory

```
void reverse(int *left, int *right) {
  if (left < right) {
    reverse(left + 1, right - 1); // reverse middle
    int temp = *left;
    *left = *right; // swap first/last elements
    *right = temp;
  }
}
```

Lots of memory

```
void reverse(int *left, int *right) {
  if (left < right) {
    int temp = *left;
    *left = *right; // swap first/last elements
    *right = temp;
    reverse(left + 1, right - 1); // reverse middle
  }
}
```

This version is tail recursive, because the recursion comes at the end. But why do we care?

lobster.eecs.umich.edu ("L18.7_reverse_tail") 6/7/2022

① tail recursion 的写法和优势.

24 Tail Recursion

- Consider these two implementations of reverse:

与for loop相比，recursion会占用o(n)个stack钟的空间。

```
void reverse(int *left, int *right) {
  if (left < right) {
    reverse(left + 1, right - 1); // reverse middle
    int temp = *left;
    *left = *right; // swap first/last elements
    *right = temp;
  }
}
```

space O(n)

然而，如果将recursion的式子放在function的结尾处，计算机optimization就会优化变成只占用o(1) constant stack space.

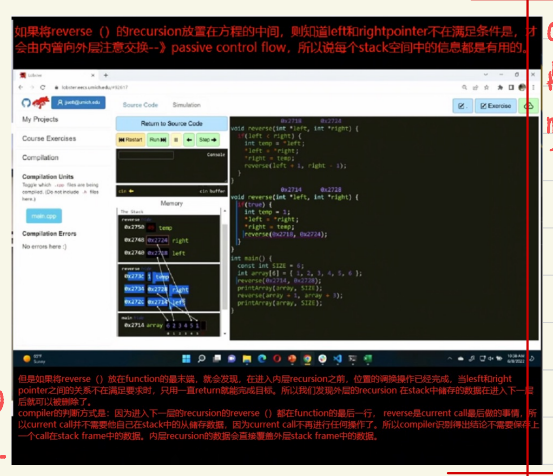
```
void reverse(int *left, int *right) {
  if (left < right) {
    int temp = *left;
    *left = *right; // swap first/last elements
    *right = temp;
    reverse(left + 1, right - 1); // reverse middle
  }
}
```

space O(1)

This version is tail recursive, because the recursion comes at the end. But why do we care?

lobster.eecs.umich.edu ("L18.7_reverse_tail") 6/8/2022

如果将reverse（）的recursion放置在方程的中间，则知道left和rightpointer不在满足条件是，才会由内省向外层主意交换-->》passive control flow，所以说每个stack空间中的信息都是有用的。



但是如果将reverse（）放在function的最末端，就会发现，在进入内层recursion之前，位置的调换操作已经完成，当left和right pointer之间的关系不在满足要求时，只用一直return就能完成目标，所以我们发现外层的recursion 在stack中储存的数据在进入下一层后就可以被删除了。
compiler的判断方式是：因为进入下一层的recursion的reverse（）都在function的最后一行，reverse是current call最后做的事情，所以current call并不需要自己在stack中的从储存数据，因为current call不再进行任何操作了，所以compiler识别得出结论不需要保存上一个call在stack frame中的数据。内层recursion的数据会直接覆盖外层stack frame中的数据。

① tail recursion 的写法和优势. const memory 的原因.

26 Tail Call Optimization (TCO)

- A function call is a **tail call** if it is the very last thing in its containing function.
- The calling function has no pending work to do after a tail call (in the passive flow), **so its stack frame isn't needed anymore.**
- Some compilers are able to recognize tail calls and optimize them.
  - Just overlay the new stack frame over the memory used for the old one.
  - In g++, -O2 includes TCO
- TCO usually has a much bigger impact for recursive functions.
  - Why?

"O" 是optimization的简写，数字 "2" 是optimization的等级。2包含了TCO。你需要告诉你的compiler你需要优化。

6/8/2022

① 如何告诉计算机优化 tail recursion

4 Recursive Factorial

这里的recursion虽然在function的最末尾，但是不能叫做tail recursion，因为在经行 "*" 操作之前，要先完成内部的recursion，一直传递到base condition

- Can we do it without a loop, using **recursion** instead?

```
// REQUIRES: n >= 0
// EFFECTS: computes and returns n!
int fact(int n) {
  if (n == 0) {
    return 1;
  }
  else {
    return n * fact(n - 1);
  }
}
```

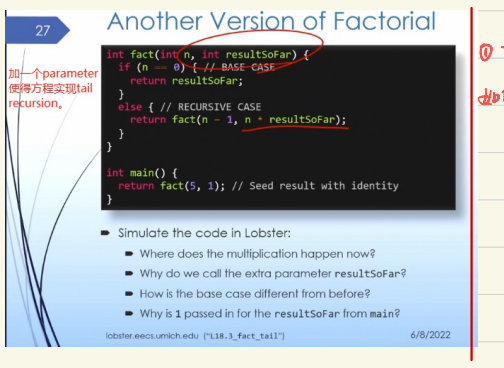
A recursive function calls itself to implement repetition.

L18.2_fact on Lobster

- Does this work?
  - How does one call set the parameters for the next?
  - How does the recursion know to stop?
  - Where does the multiplication happen?

6/8/2022

① 不是 recursion 也调在最后就叫 tail…



加一个parameter 使得方程实现tail recursion。

① factorial 如何实现 tail recursion.